Remote Vehicle Interface (RVI)

An Open Framework for the Connected Car

Rudolf Streif – Jaguar Land Rover

© 2015 Jaguar Land Rover – This presentation contains only public information.

Jaguar Land Rover (JLR)

JLR is the UK's largest automotive business, built around two iconic British car brands:

- Jaguar – one of the world's premier luxury sports car brands
- Land Rover – the world's leading manufacturer of all-wheel drive vehicles

JLR Open Software Technology Center (OSTC)

- R&D facility in Portland/Oregon focusing on next-generation technologies
- 15,000 sqft site opened in 2014
- 25,000 sqft site for expansion and incubator to open in 2015
- Currently 35 engineers and designers, grow to 80+ by the end of 2015

We want to become the best software company to buy a car from!

Where we are coming from

- **80% of connected vehicle functionality shared across platforms**
  Core IVI and Server functionality are similar, regardless of vendor. The final 20% are the services that defines the user experience.

- **A shared, open source platform will benefit OEMs**
  A joint architecture and reference implementation allows OEM to minimize cost, vendor dependencies, and security risks, letting them focus on applications and services that make a difference.

- **A shared, open source platform will benefit service providers**
  A common architecture allows service providers to easily port their products to additional OEMs, thus giving them a wider revenue stream from multiple vendors.

What we are doing

The Problem

How do we allow an in-vehicle application to exchange data with other devices and servers in a simple, robust and secure manner, regardless of connectivity?

The Solution

Build open source technology to handle authentication, authorization, discovery, and data exchange between services in a sparsely connected P2P network.

The Goal

RVI is accepted by the automotive industry as a secure, reliable, and proven technology choice for connected vehicle projects.

Benefits

- **Enable new breed of 3rd party service providers**
  Open source implementation enables start-ups to develop in-vehicle apps and their corresponding backend services, and showcase the finished product to OEMs.

- **Alleviate vendor dependency**
  OEM can replace components at will, using either external or internal resources. All IP of the replaced components belongs to the OEM.

- **Wider talent pool**
  Large competence base provided through open source community, OEMs, app developers, and professional service vendors.

Feature Set

- **Peer-to-peer based**
  Two nodes can exchange services without an internet connection.

- **Provisioning**
  Services and nodes can be added and deleted from the system.

- **Authentication & authorization**
  All executed services are authenticated and authorized against certified credentials.

- **Service discovery**
  Services and applications can discover and invoke other services.

- **Invocation**
  Services can be remotely invoked over a sparsely connected network.

Three Core Use Cases

Software/Firmware
over the Air Updates

Remote Interaction

Data Collection

HVAC Control Architecture

1. **Mobile Application Sends Command**
   HVAC App sends an message, targeting a given service URI, to Service Edge.
2. **Locate Target Node**
   Service Edge asks local service Service Discovery to resolve service name to a network address.
3. **Return Network Address**
   Specifies where the target service can be reached.
4. **Send Request to Vehicle**
   The vehicle data router processes the command.
5. **Forward Request to HVAC Service**
   The HVAC Service in the vehicle executes the command.

SOTA Architecture

1. **Web Backend Notifies Vehicle of Update**
   Web Backend sends update notification for a software package.
2. **Locate Target Node**
   Service Edge asks local service Service Discovery to resolve service name to a network address.
3. **Return Network Address**
   Specifies where the target service can be reached.
4. **Send Request to Vehicle**
   The vehicle data router processes the command.
5. **Forward Request to SOTA Service**
   The SOTA Service in the vehicle shows update details on the screen and waits for user confirmation.
6. **Vehicle Response to SOTA Server**
   Response from vehicle gets relayed to the SOTA server which then starts sending the software package or aborts the transaction based on the response.
7. **SOTA Server Sends Software Package to Vehicle in Chunks**
   The software package is split up into chunks and sent to the vehicle. SOTA Service assembles the chunks into the package.
8. **SOTA Service Installs Package**
   SOTA Service installs package and reports installation status back to the SOTA Server.

Big Data Architecture

1. **Web Backend Notifies Vehicle of Data Subscription**
   Web Backend sends a data subscription message to the vehicle.
2. **Locate target node**
   Service Edge asks local service Service Discovery to resolve service name to a network address.
3. **Return network address**
   Specifies where the target service can be reached.
4. **Send request to Vehicle**
   The vehicle data router processes the command.
5. **Forward request to Big Data Service**
   The SOTA Service in the vehicle shows update details on the screen and waits for user confirmation.
6. **Vehicle Sends Data to Big Data Servers**
   The vehicle collects the subscribed-to data and sends it to the Big Data Server in regular intervals. If the server cannot be reached the data gets cached.
7. **Big Data Server Receives Data**
   The Big Data Server receives the data and stores it in the Backend Database.
8. **Big Data Server Provides Realtime Update to Web Backend**
   The Big Data Server sends data updates in realtime to the web clients who subscribed to the data through the web backend.

Big Data Use Cases

Vehicle Tracking Information and Animation

- Show vehicle locations and travel
- Filter by date and time
- Display vehicle status information for every waypoint
- Marker moves along traveled path
- Real-time tracking

http://rvi1.nginfotpdx.net:8000

Big Data Processing

RVI Architecture

Architecture - Overview

- **API based**
  The API is the driving technology. Implementation is secondary.

- **Data Router commonality**
  Data Router connects all services on all devices.

- **Mix of open and closed source**
  Components can be off the shelf, OSS, proprietary, or a combination of the above.

- **Network complexity shielding**
  A clean transaction API alleviates services and applications from connectivity concerns.

Unified Software Stacks

Architecture – Data Router

- **Service Edge**
  Handles all traffic toward locally connected services.

- **Authorization**
  Handles certificates and authorization for all traffic.

- **Schedule**
  Handles traffic store and forward for unavailable destinations.

- **Data Link**
  Controls communication channels to other node.

- **Service Discovery**
  Identifies and locates local and remote services.

- **Protocol**
  Encodes and transmits traffic to other nodes.

JLR/AGL-developed component
OEM/3rd party proprietary component
3rd party reference component
Off the shelf component

Architecture – Backend Server

- **Data Router**
  Standard deployment.

- **Provisioning**
  Creates and distributes certificates granting access rights to nodes.

- **SOTA server**
  Manages and distributes software packages to nodes.

JLR/AGL-developed component
3rd party reference component
OEM/3rd party proprietary component

Services

Services – Requirements

- **Global namespace for all services on all nodes, worldwide**
  All services on all provisioned devices must be addressable through a single schema.

- **Localized service discovery**
  Locally connected nodes must be able to discover each other's services without Internet access.

- **Zero configuration**
  No configuration outside authorization shall be needed for a newly deployed node to join the system.

- **Network agnostic**
  A service shall be accessed the same way, regardless of the communication method used.

Services – Addressing

- **Single name space for all services**
  New services can be addressed by creating a unique name for them.

- **Service name identifies hosting node**
  Each service name, being unique across the system, carries enough information for Service Discovery to identify where the node can be found.

- **Hide network complexity**
  All service interaction with other services are done through the service name space, allowing the actual communication to be carried out behind the scenes.

Services – Service Name Example

#	Name	Description
1	Organization	Specifies a sub-section hosted by a specific entity
2	VIN Sub-tree	Specifies sub section for all vehicles
3	VIN	Vehicle Identification Number
4	Service Name	Name of the addressed service
5	Command	Command supported by the sevice

Services - Routing

1. **Application sends command**
   HVAC App sends an message, targeting a given service URI, to Service Edge.

2. **Locate target node**
   Service Edge asks local service Service Discovery to resolve service name to a network address.

3. **Return network address**
   Specifies where the target service can be reached.

4. **Send request to Vehicle**
   The vehicle data router processes the command.

5. **Forward request to HVAC Service**
   The HVAC Service in the vehicle executes the command.

Authorization

Authorization – Overview

- **Certificate based**
  Certificates, signed by a trusted provisioning server, grants node access to services.

- **Self-carried authorization**
  A node presents its certificates to another node to access its services, without provisioning server connection.

- **Service – service specific certificates**
  A certificate authorizes a specific set of services to access another specific set of services, and cannot be used outside that context.

Authorization – Use Case

1. **Create and sign certificate**
   A certificate granting access to the mobile device is created and signed with provisioning server's private key.
2. **Distribute certificate to mobile device**
   The targeted device receives its certificate
3. **Send request and certificate to Vehicle**
   The certificate states that mobile device has the right to execute the given request
4. **Validate credentials**
   The certificate and request is validated by the vehicle through a root certificate
5. **Execute request**
   The validated command is forwarded to the target service for execution

Authorization – Examples

Access List Format

`[organization]/[path]` + wildcards

`jaguarlandrover.com/vin/sajwa71b37sh1839/body/lock`

Specifies a specific vehicle's lock command in the body service.

`jaguarlandrover.com/vin/*/media/volume`

Specifies the volume control command of the media service on all vehicles.

`jaguarlandrover.com/cloud/vehicle_tracking/*`

Specifies all commands under the vehicle_tracking service.

Authorization – Topics not covered

- **Protection of certificate inside a node**
  A credential received by the mobile device needs to be secured in accordance with the mobile device/IVI/server platform

- **Certificate – device binding**
  A stolen certificate can be presented by another device to gain service access. Device binding is done on an implementation level using hardware-specific mechanisms

- **Secure communication**
  Protocol implementations are responsible for securing data transmission between nodes using SSL/TLS or similar technologies

Conclusion

- **Connected Vehicle architecture for next generation services**
- **Open source design, specification, and reference implementation**
- **Benefits the whole industry**

Thank You

Magnus Feuer

System Architect – Open Software Initiative

mfeuer@jaguarlandrover.com

Rudolf Streif

System Architect – Open Software Initiative

rstreif@jaguarlandrover.com

http://github.com/PDXostc

automotive-eg-rvi@lists.linuxfoundation.org

#automotive (IRC)